UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/003,376	11/15/2001	John P.E. Tobin	2319P	4698

57580 7590 08/29/2006

STRATEGIC PATENT GROUP, P.C.
P.O. BOX 1329
MOUNTAIN VIEW, CA 94042

EXAMINER	
VU, TUAN A	

ART UNIT	PAPER NUMBER
2193	

DATE MAILED: 08/29/2006

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	10/003,376	TOBIN, JOHN P.E.	
	Examiner	Art Unit	
	Tuan A. Vu	2193	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *09 August 2006*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-69* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-69* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

DETAILED ACTION

1. This action is responsive to the application filed 8/9/2006.

As indicated claims 1-55, and 69 have been amended. Claims 1-69 have been submitted for examination.

Claim Objections

2. Claim 69 is objected to because of the following informalities: after the recited 'obfuscated program comprising', there must be a ":" (colon) as opposed to the currently used ',' (comma), which is improper. Appropriate correction is required.

Claim Rejections - 35 USC § 101

3. 35 U.S.C. 101 reads as follows:

> Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

4. Claims 1-15, and 55-68 rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter.

> The Federal Circuit has recently applied the practical application test in determining whether the claimed subject matter is statutory under 35 U.S.C. § 101. The practical application test requires that a " useful, concrete, and tangible result" be accomplished. An "abstract idea" when practically applied is eligible for a patent. As a consequence, an invention, which is eligible for patenting under 35 U.S.C. § 101, is in the "useful arts" when it is a machine, manufacture, process or composition of matter, which produces a concrete, tangible, and useful result. The test for practical application is thus to determine whether the claimed invention produces a "useful, concrete and tangible result".

Specifically, claim 1 recites a method for increasing security, for a program executed in a system with 2 modes of execution, and comprising identifying critical code segments, executing non-critical portions in a first mode; and executing the critical code segments in another mode within some exception handlers. As a whole, the claim amounts to software execution of 2 sets

of portions using a mode for each set; and as such, does not convey that action for transforming data is reasonably materializing into result, or that data being transformed from the program execution actions are being realized into yielding a real-world application result, such that the this application result is concrete, tangible and useful. The execution steps as recited amount to actions that stay internal to the process of executing, and data being thus internal cannot be perceived as real-world tangible result, or that such result is being useful. The claim remains a non-practical application; and is rejected for leading to non-statutory subject matter.

Claims 2-14 for reciting code instrumentation/implementation leading to the execution of exception handlers, do not sufficiently convey materialization (as opposed to remaining in an internal execution and abstract state) from the above program execution into a real-world concrete, useful and tangible result; and are also rejected for the same deficiency as the base claim.

As an example of useful application result, the user of some debugger application executing the program would be displayed results from the user-level code debug only, whereas the hidden code results would be stored in a log file; i.e. a tangible result required for statutory subject matter.

Claim 15 recites partitioning software program into segments, encapsulating critical segments in handlers, and executing those segments in the 2 modes as those recited in claims 1-14. There is no reasonable teaching that this execution of some segments being partitioned enable the realization of a tangible and useful result consequential to such execution; that is, as a whole, the claim is not conveying that the execution step would lead to a form of output required

from the Practical Application Test, a result that can be construed as useful and tangible to the real world that interfaces with the recited execution.

Likewise, claims 55-68, reciting the same limitations as the above claims, thus amount to leading to a non-practical application, are also rejected as non-statutory subject matter.

Claim Rejections - 35 USC § 103

5. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

6. Claims 1-7, 15-18, 26, 36-38, 55-61, and 69 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kuzara et al., USPN: 5,450,586 (hereinafter Kuzara) and further in view of Held, USPN: 5,889,988 (hereinafter Held).

As per claim 1, Kuzara discloses a method for increasing security of a software program by obfuscation of program execution flow, wherein the software program is executed on a computer system that includes a user-level protected mode and a kernel-level unprotected mode (Figs. 9-10), the method comprising the steps of:

identifying critical code segments (e.g. *code marker, routine 1010, task switch* – col. 16, lines 19-48) to be hidden in the software program (e.g. col. 5, lines 30-49; col. 15, lines 30-39 – Note: *black box* reads on kernel execution being hidden);

executing non-critical portions of the software program in the user-level protected mode (e.g. *user code 900* – Fig. 10), wherein execution path of the non-critical portions is visible to a

debugger program executing the user-level mode (Note: user level reads on debugger program executing visible path);

executing the critical code segments (e.g. *kernel 902* - col. 16, lines 38-51), in the kernel level mode, wherein execution path of said critical segments is hidden from the debugger program executing the user-level mode (Note: the use of kernel execution --real-time "black box" --code being separated or hidden -- from the user code in the Kuzara's debug and code marker insertion --e.g. col. 5, lines 30-49-- discloses a concept of hiding execution path from the user level code debug main process).

But Kuzara does not explicitly mention about executing critical code segments within respective exception handlers; however teaches about special entry/exit points related to the use of markers to address low intrusion process for task switching and exception occurrences (e.g. *task switching ... exception event* -col. 5, lines 6-11). Analogous to using markers to support low intrusion task switching into a debug kernel execution as by Kuzara (see: task ...entered ...exited - col. 16, lines 19-48) and for use in addressing visible critical points (see Kuzara: col. 4, lines 31-42; col. 5 lines 4-13) or exceptional events as mentioned above, Held, in a debugger of embedded system, uses predefined breakpoint as well as detecting and executing of such breakpoints in a kernel debugger via exception handlers (step 807 – Fig. 8). In light of the known concept to use special program handler to address interrupt or exceptional/critical events especially in a debugging endeavor as approached by both Kuzara and Held, it would have been obvious for one skill in the art at the time the invention was made to implement the marker code for exception event or task switching by Kuzara so that it be encompassed in a interrupt/exception handler because the use of underlying handler in case of interruption or

special exception as suggested by Kuzara during user code execution would enable the underlying and readily available lower system code (kernel level) to expediently and directly address issues being at the root of such exception (see Held: col. 11, line 61 to col. 12, line 5), notably when Kuzara has intended for the debugger to operate in user code transparently from kernel black box, i.e. handling of issues without immediate support/intervening from the user 's level application.

As per claim 2, Kuzara does not explicitly disclose inserting an exception set-up handler into the software program that sets-up the exception handlers so that the exception handlers will be invoked during program execution. It is recognized that Kuzara's code marking via use of dynamic insertion (Fig. 4) or a library code setup would allow kernel code (see Figs. 9-10) to help set up task switching or handling of exceptional events as mentioned above. In view of Kuzara's above teaching as well as the user code to pass parameters to allow the OS to mark entry and exit points (col. 15, lines 41-53; col. 16, lines 7-15) and the rationale as to why Kuzara's code marking would be operated within interruption handler as set forth from above, the above code insertion to set up marking by Kuzara so that it set up kernel code handlers would have made the above claimed limitation obvious for the same reasons as set forth above.

As per claim 3, combined with Held's teaching, Kuzara also teaches insertion of code based on extraction specific parts of source code (e.g. col. 11, lines 5-9); hence Kuzara has disclosed inserting an in-line code segment in the software program, wherein the in-line code segment sets up and executes the exception set-up handler during program execution; the rationale to set up marker so that marker evokes kernel execution via use of exception handler being set forth above in claim 1.

As per claim 4, in view of the rationale for obviousness using the teaching by Held, Kuzara further includes the step of inserting a kernel level driver (re claim 1 - Note: a kernel routine to set up and evoke handler by Held reads on driver in a OS kernel) in the software program that sets-up the exception handlers so that the exception handlers will be invoked during program execution.

As per claim 5, Kuzara discloses insertion of code in the program source code to set up code marker; and kernel code to execute point at which code switch being supported with handlers is being deemed obvious as set forth in claim 1 or 4 above. Hence, Kuzara in view of Held discloses the step of inserting an in-line code segment in the software program, wherein the inline code segment invokes the kernel-level driver, thus causing the exception handlers to be invoked during program execution.

As per claim 6, Kuzara in view of Held disclose using the set-up handler to initiate a set-up of breakpoint or marker for code switch such that the critical code segments in the exception handlers are executed at appropriate times during the program execution (re claim 1) but Kuzara does not explicitly disclose debug registers. Held discloses using code switch and set up debug registers along with breakpoints (e.g. col. 10, lines 11-28). Based on the rationale to segregate user code and kernel level tasks switching as intended by Kuzara when handling area of code at which critical points need to be handled more particularly, it would have been obvious for one of ordinary skill in the art at the time the invention was made to implement the task switch and breakpoints shown by Kuzara with the use of dedicated hardware registers as shown by Held because according to Held, this would dedicate special hardware resource per context of a task

switcher as taught by Kuzara hence obviate or prevent the main processor for being held permanently because of some potentially ill-behaved tasks.

As per claim 7, Kuzara teaches insertion of code in the program source to set up marking of code (re claim 3); hence has disclosed executing the in-line code segment prior to a point in the program flow where any of the critical code segments was removed for placement into the exception handlers (e.g. col. 5 lines 4-13 - Note: after the marking is done, code markers is only detected later in order to address critical area of code).

As per claim 15, Kuzara discloses a method for increasing security of a software program by obfuscation of the software program execution flow, the method comprising the steps of:

partitioning the software program into code segments, each code segment having a particular location within the program execution flow (e.g. Fig. 7);

identifying one or more critical code segments (e.g. *code marker, routine 1010, task switch* – col. 16, lines 19-48);

encapsulating the critical code segments in respective kernel code execution (col. 5, lines 30-49; col. 15, lines 30-39 – Note: *black box* reads on kernel execution being segments of code being encapsulated away from the user's code); and

during software program execution, executing code segments in a user-level mode in their respective relative location in the execution flow path (e.g. *code marker, routine 1010, task switch* – col. 16, lines 19-51), wherein the execution path is visible to a debugger program executing the user-level mode(Note: user level reads on debugger program executing visible path);

and executing the code segments containing the critical code segments in their respective relative location within the execution path, within the program execution flow in the kernel mode (e.g. *kernel 902* - col. 16, lines 38-51; col. 15, lines 30-39), wherein the execution path is hidden from the debugger program executing the user-level mode (Note: the use of kernel execution -- real-time "black box" --code being separated or hidden -- from the user code in the Kuzara's debug and code marker insertion --e.g. col. 5, lines 30-49-- discloses a concept of hiding execution path from the user level code debug main process)

But Kuzara does not explicitly disclose encapsulating in exception handlers; nor does Kuzara disclose that the kernel-executed segments are exception handlers, i.e. the exception handler containing the critical code segments. But this exception handler limitation as to implementing the kernel critical segment execution under an exception handler has been addressed in claim 1 above.

As per claim 16, Kuzara discloses a method for obfuscation of computer program execution flow to increase computer program security, the method comprising the steps of:

breaking the computer code into a plurality of code segments (Fig. 7), and identifying critical code segments to be obfuscated (col. 5, lines 30-49; col. 15, lines 30-39 – Note: *black box* reads on segments being handled by kernel execution, i.e. being obfuscated);

embedding each of one or more critical code segments within respective first kernel execution so that when executed, the critical code segments are executed in the kernel mode (e.g. *kernel 902* - col. 16, lines 38-51; col. 15, lines 30-39), wherein the execution path of the critical code segments is hidden from the debugger program executing in the user-mode (Note: the use of kernel execution --real-time "black box" --code being separated or hidden -- from the user

code in the Kuzara's debug and code marker insertion --e.g. col. 5, lines 30-49-- discloses a concept of hiding execution path from the user level code debug main process);

embedding an in-line code segment within a first one of the remaining plurality of code segments for setting up and invoking the kernel execution of critical segments (e.g. col. 11, lines 5-9);

wherein when executed, the remaining plurality of code segments are executed in the user-level mode, the execution path thereof being visible to the debugger program executing in said user-level mode (e.g. col. 16, lines 38-51- Note: entering and exiting kernel areas reads on execution path visible by debugger prior to entering kernel marked areas).

But Kuzara does not explicitly disclose that first the kernel-executed critical code segments are execution of segments within exception handlers, nor does Kuzara disclose providing an exception set-up handler to set up operation of the first exception handlers. But this limitation as to implementing the kernel critical segment execution under an exception handler has been addressed in claim 1 above.

As per claim 17, the limitation as to provide set-up handler to initiate a set-up of debug registers, such that the critical code segments in the exception handlers are executed at appropriate times during the program execution, correspond to the subject matter of claim 6; hence will incorporate the rationale of rejection as set forth therein.

As per claim 18, this claim corresponds to claim 7; hence is rejected using the rationale as set forth therein.

As per claim 26, Kuzara does not explicitly disclose providing entry code for the first exception handlers to determine a nature of the exception; but Kuzara disclose various events

called deemed critical to evoke exception handlers (col. 4, line 42 to col. 5, line 11). In view of the rationale to impart exception handler inside critical kernel code execution as set forth in claim 1, and the nature of what type of events to address from Kuzara (see step 604 - Fig. 6). In view of a specific handler as taught from Held's kernel execution (e.g. handler supports task specific – col. 7, lines 23-26; Fig. 6), so to support the correct type of exception or debug task being passed to a kernel handler, it would have been obvious for one skill in the art at the time the invention was made to implement the exception handlers and kernel execution thus mentioned above to address debug events as taught by Kuzara so that a provision is implemented by means suggested by Held to evoke the correct handlers, i.e. a code to identify the nature of the event or type of exception as raised above by Kuzara, because evoking of a specific handler would depend on such identification code, such as recognized in Held's teaching.

As per claim 36, Kuzara discloses a method for obfuscation of computer program execution flow to increase computer program security, the method comprising the steps of:

partitioning the program into a plurality of non-critical code segments and at least one critical code segment (e.g. Fig. 7);

obfuscating the critical code segments by embedding each of one of the critical code segments within a kernel execution (e.g. *kernel 902* - col. 16, lines 38-51), wherein when executed the execution path of said critical code segments is hidden from the debugger program executing the user-level mode (Note: the use of kernel execution --real-time "black box" --code being separated or hidden-- from the user code in the Kuzara's debug and code marker insertion --e.g. col. 5, lines 30-49-- discloses a concept of hiding execution path from the user level code debug main process).;

providing a driver to set up operation of the first kernel execution (e.g. col. 11, lines 5-9); and

embedding an in-line code segment within a first non-critical code segment for invoking the driver when the program is executed (e.g. *code marker, routine 1010, task switch* – col. 16, lines 19-51), wherein when executed, the remaining plurality of code segments are executed in the user-level mode, the execution path thereof being visible to the debugger program executing in said user-level mode (e.g. col. 16, lines 38-51- Note: entering and exiting kernel areas reads on execution path visible by debugger prior to entering kernel marked areas).

Also as mentioned in claim 1, Kuzara does not explicitly teach segments within kernel execution are segments within exception handlers; nor does Kuzara teach setup operation of a first exception handler; however, this exception handler limitation has also been addressed using the rationale as set forth in claim 1.

As per claims 37-38, these claims correspond to claims 6-7, respectively; hence is rejected using the rationale as set forth therein, respectively.

As per claim 55, Kuzara discloses a computer-readable medium containing a software program that is to be executed on a computer system that includes a user-level protected mode and a kernel-level unprotected mode (Figs. 9-10), wherein the software program contains critical code segments to be obfuscated during program execution flow to increase security of the software program, the software program comprising instructions for:

executing (non-critical portions) of the software program in the user-level protected mode, wherein execution path of the non-critical portions is visible to a debugger program executing the user-level mode; and

executing (the critical code segments) wherein execution path of said critical segments is hidden from the debugger program executing the user-level mode;

all of which executing steps having been addressed in claim 1.

Also as mentioned in claim 1, Kuzara does not explicitly teach execution of critical segments within respective exception handlers; thus, this limitation has also been addressed using the rationale as set forth in claim 1.

As per claims 56-61, these claims correspond to claims 2-7, respectively; hence are rejected using the corresponding rationale as set forth therein.

As per claim 69, Kuzara discloses a system for obfuscation of program execution flow, comprising:

a computer system including a processor, memory, an operating system that provides kernel-level and user-level execution modes (Fig. 1, 5), and debug resources to support the generation and processing of exceptions at specified addresses (Note: this exception at specified address is inherent to any computer exceptions that are being handled); and

an obfuscated program comprising:

a plurality of non-critical code segments, which when executed, are executed in the user-level mode, wherein execution path of the non-critical portions is visible to a debugger program executing the user-level mode (e.g. *code marker, routine 1010, task switch* – col. 16, lines 19-51);

a plurality of critical code segments encapsulated within respective kernel execution segments, wherein execution path of said critical segments is hidden from the debugger program executing the user-level mode (e.g. *kernel 902* - col. 16, lines 38-51 Note: the use of kernel

execution --real-time "black box" --code being separated --or hidden -- from the user code in the Kuzara's debug and code marker insertion --e.g. col. 5, lines 30-49-- discloses a concept of hiding execution path from the user level code debug main process); and

a set-up handler for setting up the kernel execution (e.g. col. 5, lines 30-49; col. 15, lines 30-39) containing the critical code segments are executed in the kernel level mode (*kernel 902* - col. 16, lines 38-51).

But Kuzara does not discloses that kernel execution segments are exception handlers; nor does Kuzara explicitly discloses a setup code for setting debug registers such that the exception handlers containing the critical code segments are executed in the kernel mode. But these limitations have been addressed respectively in claim 1 and 17.

7. Claims 8-14, 19-25, 27, 39-54, and 62-68 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kuzara et al., USPN: 5,450,586 and Held, USPN: 5,889,988 and further in view of Cardoza et al., USPN: 5,630,049 (hereinafter Cardoza)

As per claim 8, Kuzara discloses including in-line code segment to install the exception set-up handler for a current thread (re claim 3, in view of Held as set forth in claim 1), such that when an exception occurs, the operating system hands control to the exception set-up handler for execution; however, Kuzara does not mention about on an exception handler linked list even though teaches about breakpoints or markers being stored in a table (table *1002* –Fig. 10). The setting of breakpoints so that it follows a list a table or a chained list like a linked list is further disclosed by Cardoza. Analogous to the breakpoints handling by Held in light of Kuzara, Cardoza, in a debug system where user debug commands are generated from the host computer to a embedded device for set up breakpoints and wherein breakpoints are handled via message at

the target system, and further discloses a linked list to maintain user's generated breakpoints (e.g. col. 26, lines 29-60). The use of list or table in order to dynamically support/establish points at which the kernel code is invoked to establish exception handling switches as mentioned in claim 1 would be recognized via Kuzara's teaching; and in view of such, it would have been obvious for one of ordinary skill in the art at the time the invention was made to maintain such list or table in terms of a linked list as taught by Cardoza, because this would enable Kuzara's user-activated code marking setup call to be informed on the dynamic adding or deleting of breakpoints for a particular OS thread, and this is fostered via Cardoza's linked list and its usefulness based on known programming language usage of such construct (see Cardoza: col 26, lines 53-60)

As per claims 9 and 10, with respect to the rationale of claim 8, the use of linked list by Kuzara in view of Cardoza also includes, by virtue of a integral linked list adding/removing scheme, the removing (re claim 9) the exception set-up handler from the linked list after execution; and further includes (re claim 10) inserting each of the exception handlers into the linked list.

As per claim 11, Kuzara in view of Held discloses when an exception is raised to the operating system, handing control down to the layer of code hidden to the user code (see Kuzara: col. 5, lines 30-49; col. 15, lines 30-39). But Kuzara does not explicitly disclose handling down to the linked list until one of the exception handlers determines that the exception is one that the exception handler is designed to handle. The set up of breakpoints in terms of points at which the kernel code would handle critical points via exception handler (using Held) as set forth in claim 1, have been recognized to yield benefits from using a set up via a linked list

as set forth in claim 8; hence the limitation as to hand down to a linked list for such critical points to be addressed based on the existence of a marker or breakpoints being still alive (not yet removed) in such list would also have been obvious in light of the rationale of claim 8.

As per claim 12, in light of the linked list being traversed in order to find out which breakpoints remain as to execute under or handle over an exception handler as set forth above, Kuzara in view of Held/Cardoza discloses if the exception is one for which the current exception handler was designed to handle, executing the critical code segment included in the current exception handler.

As per claim 13, Kuzara discloses a return after a context switch and exception events (e.g. col. 16, lines 19-48; col. 5, lines 4-11) and in light of Held's exception handler with restoring of context (e.g. col. 9, lines 52 to col. 10, line 10), this context switch encompasses the step of: if the exception is one for which the exception set-up handler was designed to handle, setting a return address for the exception set-up handler when the exception processing completes. Hence, in view of the obviousness rationale as set forth in claim 1, Kuzara in view of Held has disclosed setting a return address for the exception handler when the exception processing completes.

As per claim 14, the concept of daisy chaining a sequence of exception handler falls under the linked list concept as being addressed above in claim 8; hence the claim will incorporate the same rationale as set forth therein.

As per claims 19-21, these claims correspond to claims 8-10, respectively; hence are rejected using the corresponding rationale as set forth therein.

As per claims 22-25, these claims correspond to claims 11-14, respectively; hence are rejected using the corresponding rationale as set forth therein.

As per claim 27, with respect to identifying of the nature of exception handler to use as set forth in claim 26, Kuzara does not explicitly disclose including the step of handling the exception if the nature of the exception is applicable to the first exception handler, otherwise handing exception processing to the next available exception handler. However, the selecting of a appropriate handler by searching a first handler then a second handler would also have been obvious in view of chaining the handlers as set forth in claim 8 above based on the rationale therein.

As per claims 39-45, these claims correspond to claims 21-27, respectively; hence is rejected using the rationale as set forth therein, respectively.

As per claim 46, the limitation of selecting from the plurality of code segments to obfuscate within the first exception handlers based on the value falls under the ambit of determining the nature of the exception as set forth in claim 44; and incorporates the rationale as set forth therein (Note: the limitation as to obfuscating the algorithms within the segment, i.e. code constructs in a segment, is inherent to the concept of hiding as set forth in claim 1).

As per claim 47, Kuzara (in light of Held - Note: the set up of address return from a exception handling is inherent to the Held's handler) discloses exception handlers to modify a return address (e.g. col. 18, lines 22-61; col. 16, lines 6-16) to be used after completion of the exception processing

As per claim 48, Kuzara (in light of Held) discloses rearranging non-embedded code segments out of normal execution order (col. 18, lines 22-61; col. 16, lines 6-16; Fig. 8), and

setting exception addresses and return addresses to ensure proper program sequencing to further obfuscate the program execution flow (Note: the set up of address return from a exception handling is inherent to the Held's handler).

As per claims 49 and 50, Kuzara discloses including a plurality of code segments within the first exception handler to modify exception condition (Note: set up address for exit and return reads on modifying), such segment including a plurality of code segments (re claim 3: in light of the inline code inserting as addressed in claim 3).

As per claim 51, this claim falls under the ambit of the subject matter about the nature of the exception has been addressed in claim 26; hence incorporates the rejection as set forth therein.

As per claim 52, this claim includes additional code for linking exception handlers in a chain, all of which having been addressed in claims 8 and 14; hence the claim incorporates the rationale of rejection as set forth therein.

As per claim 53, the step of setting up exceptions to occur within one or more first exception handlers, to be handled by other first exception handlers, to further obfuscate the program execution flow would be treated as this falls under the subject matter (linked list/daisy chains) of claims 8 and 14; hence would be rejected with the rejection as set forth therein.

As per claim 54, the step of repeating the embedded exceptions within exception handlers such that the required exception processing occurs at a plurality of exception processing levels, to further obfuscate the program execution flow would falls under the subject matter of a daisy chain or a linked list as set forth in claims 8 and 14; hence would be rejected with the rejection as set forth therein.

As per claims 62-68, these claims correspond to claims 8-14, respectively; hence are rejected using the corresponding rationale as set forth therein.

8. Claims 28-31 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kuzara et al., USPN: 5,450,586, and Held, USPN: 5,889,988; and further in view of Admitted prior Art (APA) and Hagimont et al., "Hidden Software Capabilities", Proceedings of the 16th International Conference on Distributed Systems, May 1996, pp. 1-14 (hereinafter Hagimont) .

As per claims 28 and 29, Kuzara discloses debugging with user code being separated from Operating System lower layer execution by using enhancement to a black box concept for hiding from a user level application, i.e. obfuscating the critical code segments to prevent user from directly accessing prohibited code segments in the kernel; but does not explicitly state that such obfuscating is to prevent reverse engineering (re claim 28) or to hide anti-piracy algorithms (re claim 29). The art of allowing user with certain privileges while hiding or inhibiting access to the user or malicious external intents was a known concept in software distributing via network's communicating with embedded device by Kuzara (col. 15, lines 20-39; Fig. 2) or Held (Fig. 4-5). Other concerns for software application protection as by Kuzara are the software piracy issues being also recognized as unwanted code modifications/hacking or access/privilege or copyright violation against proprietary application data as above noted; and such piracy is recognized via the Admitted Prior Art (APA) as mentioned in the instant Application Specifications (BACKGROUND : Pg. 1-2) and raised via the teachings by Hagimont. In an analogous manner to protect application software from being violated in the access right scheme so well known in software piracy (see APA), Hagimont, in a system to communicate client-server software capabilities or application data transfer like the embedded debug paradigm by

Kuzara, discloses protection schemes via definition language, capabilities or stubs to avoid malicious and unwanted interfering of data between the OS/kernel and the exchanged application data (see Introduction: *access rights*; Chap. 5 - *Arias ...hook ...through kernel exception*; ch. 2-5). Since reverse engineering is one form of piracy wherein software application data is observed via malicious intrusion therein in order to make unwanted modifications thereto, it would have been obvious for one skill in the art at the time the invention was made to implement the protection mode in the debugger environment by Kuzara (in view of the exception handling by Held) so that such protection would be implemented via code Kernel exception handling as mentioned through Hagimont's kernel hooks or stubs for hiding software or application data, such protection being such as to implement not only anti-piracy (as by APA) and also against reverse-engineering; because according to Hagimont the use of extension code and capabilities to hide software from such access right violation, the integration of software in development in network would be more secure and more flexible (see Introduction and chap. 2.1).

As per claim 30, Kuzara (in light of Held - Note: the set up of address return from a exception handling is inherent to the Held's handler) discloses exception handlers to modify a return address (e.g. col. 18, lines 22-61; col. 16, lines 6-16) to be used after completion of the exception processing

As per claim 31, Kuzara (in light of Held) discloses rearranging non-embedded code segments out of normal execution order (col. 18, lines 22-61; col. 16, lines 6-16; Fig. 8), and setting exception addresses and return addresses to ensure proper program sequencing to further obfuscate the program execution flow (Note: the set up of address return from a exception handling is inherent to the Held's handler).

9. Claims 32-35 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kuzara et al., USPN: 5,450,586; Held, USPN: 5,889,988; and Hagimont (in light of APA),"Hidden Software Capabilities" and further in view of Cardoza et al., USPN: 5,630,049

As per claim 32, Kuzara (in light of Held/Hagimont) discloses the step of including code handlers to modify exception conditions but does not disclose code within a first handler to modify exception conditions to support future exceptions; however, the concept to set up chained exception has been addressed using Cardoza; and in light of the daisy chain and linked list limitations being addressed in claims 8 and 14, the setup code to provide support for further exceptions would have been obvious herein for the same rationale as set forth therein.

As per claim 33, Kuzara including a plurality of code segments within the first exception handler (re claim 3: in light of the inline code inserting as addressed in claim 3).

As per claim 34, the claim falls under the ambit of the subject matter about the nature of the exception has been addressed in claim 26; hence incorporates the rejection as set forth therein.

As per claim 35, this claim falls under the ambit of the subject matter about linking exception in a chain; which has been addressed in claims 8 and 14; hence incorporates the rationale as set forth therein.

Response to Arguments

10. Applicant's arguments filed 8/9/2006 have been fully considered but they are not persuasive. Following are the Examiner's observations in regard thereto.

Rejections under USC 103(a):

(A) Applicant has submitted that none of the references cited teach or suggest combination of 'executing non-critical portions ... wherein an execution path ... visible ... user-level mode' and 'executing the critical code ... is hidden from the debugger... protected mode' as required in amended claim 1 (Appl. Rmrks, pg. 21, bottom); and that according to the present invention, while critical code segments are executed within respective Exception handlers, user-level debuggers are incapable of analyzing code within such handler, therefore unable to provide information for hackers to reverse engineer (Appl. Rmrks, pg. 23, top). The claim as amended amounts to executing some portions in a user non-kernel protected mode and others -- which are identified as critical, in a non-user kernel unprotected mode. The fact that Kuzara's code is not user-level but pertains only to a kernel mode is sufficient to teach that kernel execution thereof is hidden from the user's debugging environment, i.e. from the user-level debugger main thread. The claim does not make it more specific as to how this hiding methodology is particularly implemented except for the somewhat well-known fact that the kernel-executed portions happen to be within some handlers – Note: setting up of watch-points in a debug main flow and transferring the control thereof at which point to a external particular code to handle it with specialized tasks is known concept and that's how Kuzara's code markers are implemented. Based on the rationale of the 103(a) rejection, context switching from the front end to the kernel back end require a particular point at which code handlers are invoked to address such specially identified and usually critical spots; and this teaching is in the cited parts of Kuzara in that Kuzara sets up critical points and enable via insertion of code marker the context switch for *black box* code invocation to execute and to write results thereof to special areas (col. 5, li. 30-49). The argument in regard to the hiding so to make on visible to the main debug program

appears to base on teachings outside of the very language of the claim. Claim 1 amounts to identifying particular spots so to enable the execution thereof using some handlers; and there is no more specifics as to how this hiding feature is particularly done in order to preclude Kuzara's *black box* code execution from fulfilling this *hiding* from the user's level.

For clarification, let's take an example of a invention, and see if such invention has some patentable weight: a method for optimizing code comprising identifying spots where code can be optimized, marking them, and executing them with special handlers, the execution thereby is optimized. There is nothing being optimized about executing program code in which some spots are marked. Specific teachings has to be described in the claim in order for a crucial feature like optimization (e.g. what has been marked and what is particularly done when such marked spots are encountered) to bear some weight in light of useful arts and known prior methodology. A wishful thinking imparted in the language of the claim without any description of how such wishful endeavor has been reasonably implemented does not amount to a patent-eligible subject matter; for it is bordering on intended use. In claim 1, the *hiding* as claimed is mere target or purpose, but the way to go about *hiding* has not been reasonably recited to distinguish the claimed invention from the debug using markers and code handler by the combined teachings of Kuzara and Held, among others. If *hiding* actually amounts to calling handler code, then Kuzara's in light of Held have met the limitation.

(B) Applicant has submitted that Kuzara's debugger is only to make critical points visible (Appl. Remrks, pg. 23, bottom, pg. 24, top para) without user's knowing about the inner works of the RTOS; while the instant invention is a debugging tool that supports hiding operating system from the user. In response, it seems evident that there are no specific teachings in claim which

are explicitly enforcing that the so-called *hiding* be effectuated by means of some special methodology, notwithstanding the observation about using code handler to address special watch-points as set forth in section A above. The rejection has been set forth based on the broad reasonable interpretation of the elements as they are recited; and according to the hiding concept as recited in claim 1, for example, this concept amounts to implementation of 2 entities: (i) identifying of critical segments and (ii) executing of these within exception handlers. The rejection has pointed to where Kuzara has set up critical points at which some special markers would be intended to invoke special code execution like performing data dump or addressing exceptions events (col. 5, lines 4-25); and has used the rationale as to bring the teaching by Held to render obvious the use of code handlers to address Kuzara's task switching upon encountering markers intended for exception events. Applicant's arguments fail to comply with 37 CFR 1.111(b) because they amount to a general allegation that the claims define a patentable invention without specifically pointing out how the language of the claims patentably distinguishes them from the references.

(C) Applicant has submitted that Held, Cardoza fail to cure Kazura (Appl. Rmrks, pg. 24, middle). Sections A and B above will be referred to in order to address this argument. This argument is considered an allegation when the claimed invention is not recited in terms so to distinguish or render unobvious the prior art used; especially when the rationale of rejection has set forth clearly the reasons why the limitations would have been obvious; against which, Applicant has failed to provide clear reasons why such combined teachings would generate adverse effects, and so very specifically how. In short, the very nature of this *hiding* limitation is merely expressed in the language of the claim as a intended purpose, and is not sufficiently

conveyed therein in terms of its implementation so to preclude the combined teachings by Kuzara and Held --at the time the invention was made in view of known concept in exception handling and breakpoint-based debugging technologies— from rendering the claimed hiding implementation obvious; or to make such hiding technique more distinguishing than the implementation disclosed by the references as the references also teach identified points where exception can be addressed within special code handlers.

Therefore, the claims will stand rejected as set forth above.

Conclusion

11. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Tuan A Vu whose telephone number is (272) 272-3735. The examiner can normally be reached on 8AM-4:30PM/Mon-Fri.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Kakali Chaki can be reached on (571)272-3719.

The fax phone number for the organization where this application or proceeding is assigned is (571) 273-3735 (for non-official correspondence - please consult Examiner before using) or 571-273-8300 (for official correspondence) or redirected to customer service at 571-272-3609.

Any inquiry of a general nature or relating to the status of this application should be directed to the TC 2100 Group receptionist: 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished

applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Tuan A Vu
Patent Examiner,
Art Unit 2193
August 25, 2006